Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Übungsblatt Nr. 6

Aufgabe 1: Scheduling-Verfahren

Erklären Sie die grundlegenden Unterschiede zwischen Standard- und Echtzeitscheduling-Verfahren. Nennen Sie Beispiele für die jeweilige Klasse.

Lösung:

Beim Standard-Scheduling sind dem Scheduler überhaupt keine Informationen über eventuelle Zeitbeschränkungen einzelner Prozesse bekannt. Daher kann er weder solche Restriktionen berücksichtigen noch neue Prozesse ablehnen, die Zutritt zum System verlangen. Gewisse Vorrechte für einzelne Prozesse können nur über Prioritäten vergeben werden. Ein typisches Beispiel ist der UNIX-Time-Sharing Scheduler.

Beim Echtzeitscheduling erhält der Scheduler für jeden Prozeß eine Echtzeitrestriktion, die im wesentlichen den benötigten Prozentanteil an Prozessorzeit angibt. Da sich dieser Anteil aber auf eine bestimmte Zeitspanne bezeiht, wird die Information in Form von Periode-Berechnungszeit-Tupeln übergeben, die angeben, wieviel Berechnungszeit ein Prozeß in einer bestimmten Periode benötigt. Basierend auf diesen beiden Parametern aller Prozesse werden die Ablaufpläne (Schedules) erstellt und die Schedulability-Tests durchgeführt. Ein neuer Prozeß kann vom Scheduler abgelehnt werden, wenn er die Einhaltung der Echtzeitrestriktionen aller anderen schon laufenden Prozesse gefährdet oder unmöglich macht. Beispiele für Echtzeitscheduling-Algorithmen sind Rate-Monotonic (RM) und Earliest-Deadline First (EDF).

Aufgabe 2: Schedules

Gegeben seien drei Prozesse P_1, P_2 und P_3 mit den folgenden Anforderungen an Periode und Berechnungszeit: $p(P_1) = 50$ ms, $p(P_2) = 120$ ms, $p(P_3) = 70$ ms, $c(P_1) = 10$ ms, $c(P_2) = 40$ ms, $c(P_3)= 25$ ms. Erstellen Sie Ablaufpläne nach den folgenden Verfahren:

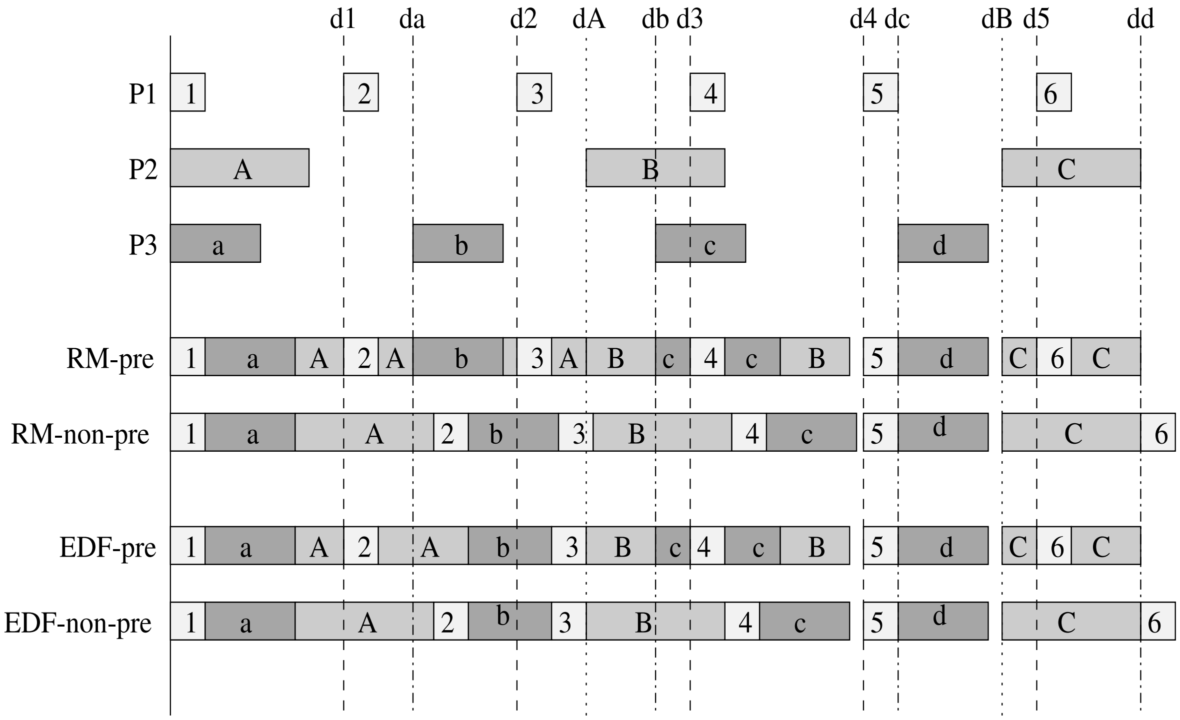
(a) Rate Monotonic preemptive

(b) Rate Monotonic non-preemptive

(c) Earliest Deadline First preemptive

(d) Earliest Deadline First non-preemptive

Lösung:



Aufgabe 3: QThreads

(a) Erläutern Sie den Unterschied zwischen einem Prozeß und einem Thread.

Lösung:

Sowohl bei Thread als auch bei Prozeß handelt es sich um Begriffe aus dem Betriebssystembereich. Ein Prozeß setzt sich zusammen aus einem Adreßraum und einem Programmzähler. Der Programmzähler bestimmt den Kontrollfluß (den Programmablauf), während der Adreßraum den Zugriff auf Code und Daten des Programmes ermöglicht. Gleichzeitig bildet der adreßraum einen Schutzwall, der von anderen Prozessen nicht durchbrochen werden kann, d.h., ein Prozeß kann nicht auf den Adreßraum eines anderen Prozesses zugreifen. Ein Thread dagegen besitzt nur einen Programmzähler, aber keinen Adreßraum. In einem Multithread-Betriebssystem kann ein Prozeß typischerweise mehrere Threads und damit mehrere Kontrollflüsse enthalten. Damit wird eine nebenläufige Abarbeitung von Instruktionen eines Programmes erzielt, wodurch oftmals eine natürlichere Modellierung und auf Parallelrechnern eine Programmbeschleunigung möglich wird. Allerdings besteht jetzt nicht mehr der Schutz, den ein Adreßraum liefert. Der Programmierer ist selbst dafür verantwortlich, daß sich einzelne Threads nicht gegenseitig fälschlicherweise Daten überschreiben. Zu diesem Zweck setzt er sogenannte Schloßvariablen, Semaphore oder Monitore ein.

(b) Was ist generell der Vorteil der Verwendung der sogenannten QThreads verglichen mit „normalen“ Echtzeitthreads?

Lösung:

QThreads sind flexibler, da sie die Angabe von Bandbreiten bei den Parametern Berechnungszeit und Periode erlauben. Damit wird einerseits der Anwender von der Last befreit, einen einzigen „richtigen“ Wert angeben zu müssen, andererseits erhält der Scheduler die Möglichkeit, Parameter nach der aktuellen Lastsituation auszuwählen.

(c) Welche Maßnahmen kann der QThread-Scheduler ergreifen, um eine Menge von QThreads auszuführen, obwohl die aktuelle Parametereinstellung das Schedulability-Kriterium nicht erfüllt?

Lösung:

Verringerung der Berechnungszeit oder Erhöhung der Periode einzelner oder mehrerer Threads

(d) Warum kann es für einen Scheduler sinnvoll sein, die QoS-Parameter seiner Threads zu variieren und nicht gleich die niedrigste Einstellung zu verwenden? Betrachten Sie die zwei Situationen, daß (a) der Scheduler dem Benutzer „gehört“ oder daß (b) der benutzer ein „Kunde“ des Schedulers ist.

Lösung:

In der ersten Situation wird der Scheduler natürlich alles tun, um den Benutzer zufrieden zu stellen, also möglichst gute Parameterwerte auswählen. In der zweiten Situation kann der Wert eines Parameters von der Bezahlung abhängen. Der Scheduler kann dem Benutzer einen besseren Wert bieten, wenn dieser bereit ist, mehr zu zahlen.